시스템 프로그래밍

Assignment2-1

Class : A

Professor : 김태석 교수님

Student ID : 2022202065

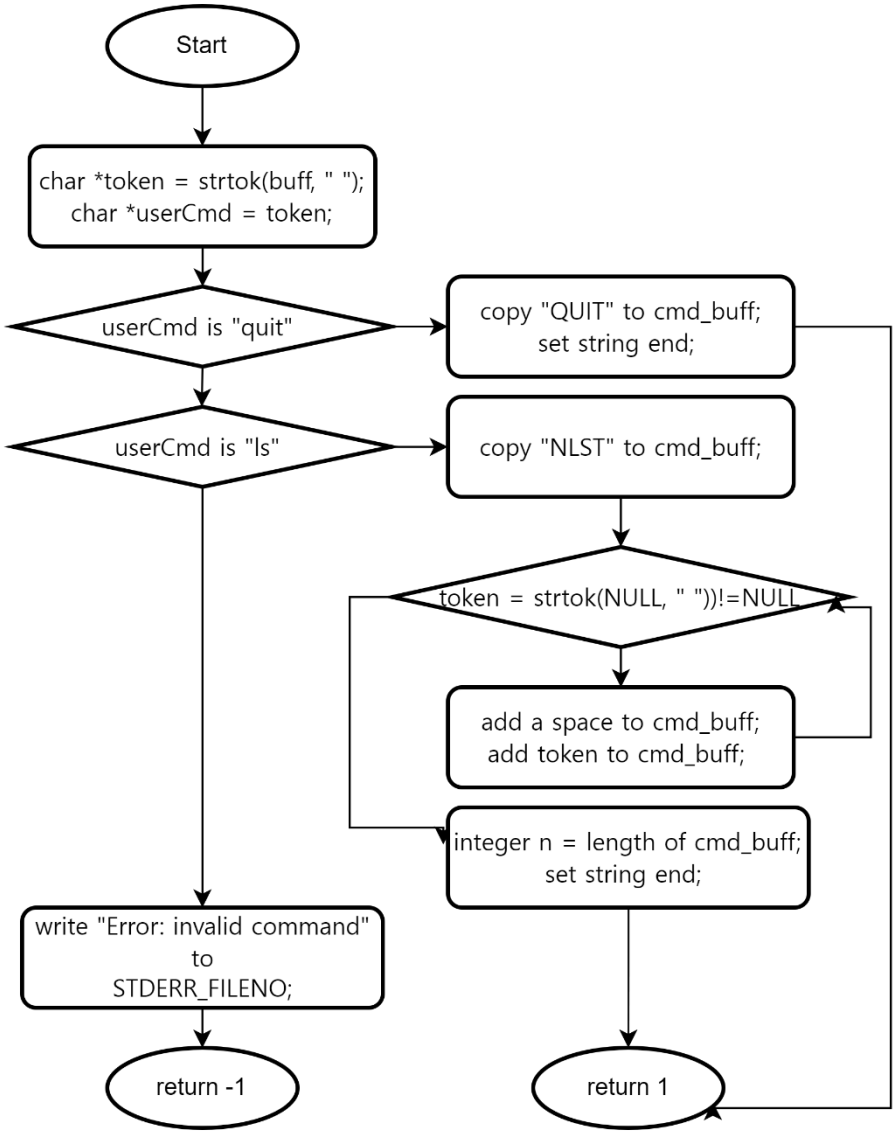
Name : 박나림

Introduction

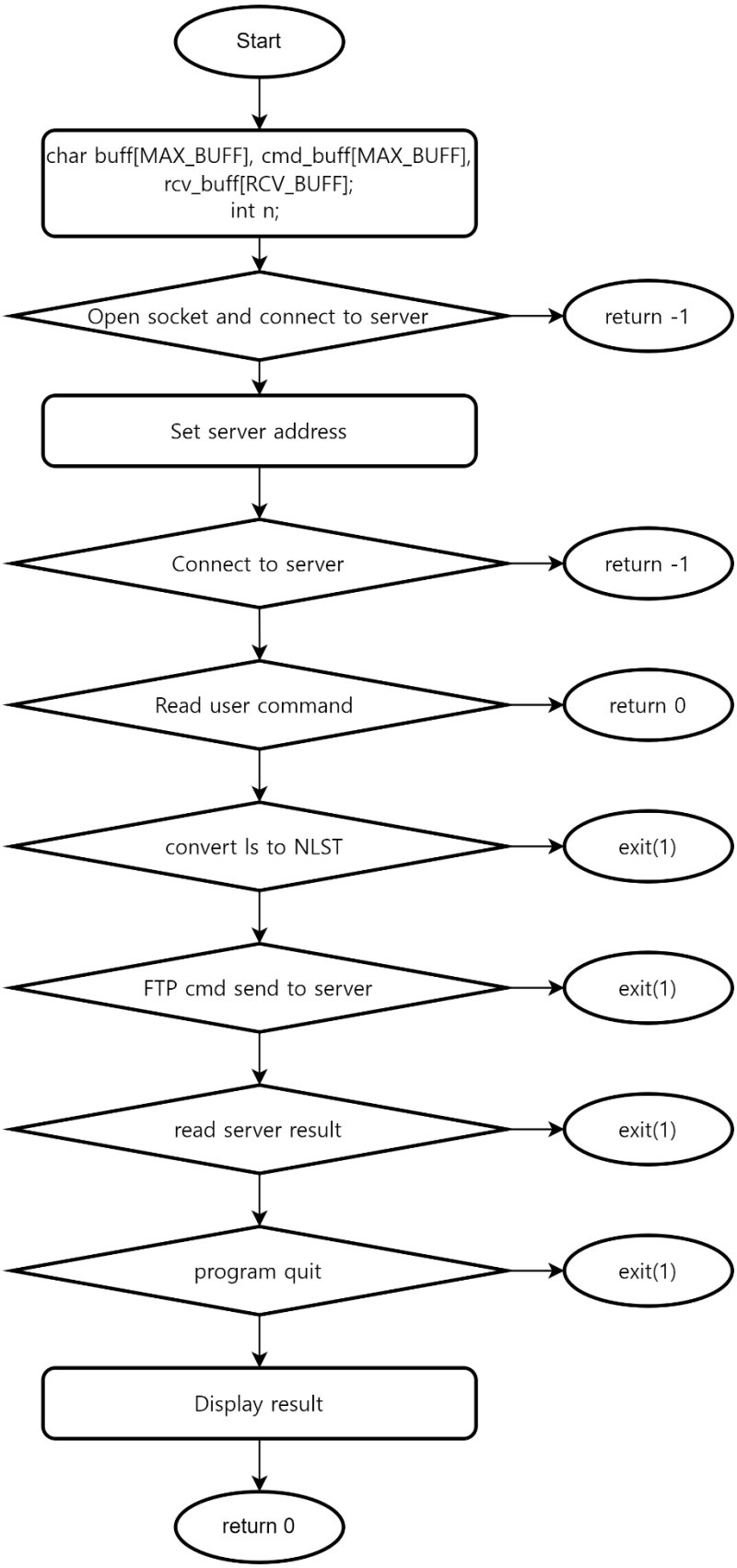
FTP 를 구현하는 프로젝트 중 소켓 프로그래밍을 하는 과제이다. 이번 프로젝트에서는 TCP server, TCP client 를 구현하여 서로 소켓을 생성해 연결하는 것이 목표이다. 서버를 먼저 실행하여 포트번호를 지정하고 listen 상태가 되면, 클라이언트를 실행하여 ip 주소와 포트를 지정하고 서버에게 연결 요청을 보낸다. 서버는 클라이언트의 ip, 포트 정보를 출력하고 클라이언트로부터 받은 ftp 명령어를 실행시킨다. 그 결과를 소켓을 통해 다시 클라이언트에게 보내면, 클라이언트는 해당 결과를 화면에 출력한다. 이러한 과정은 user 가 quit 명령어를 입력할 때까지 반복된다. 그 사이에 여러 오류들에 대한 예외처리를 하도록 한다. 이번 과제를 통해 네트워크 상에서 데이터 교환을 가능하게 해주는 소켓을 공부해보도록 한다.

Flow chart

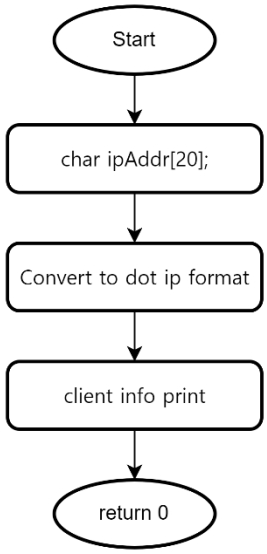
cli.c - conv_cmd()



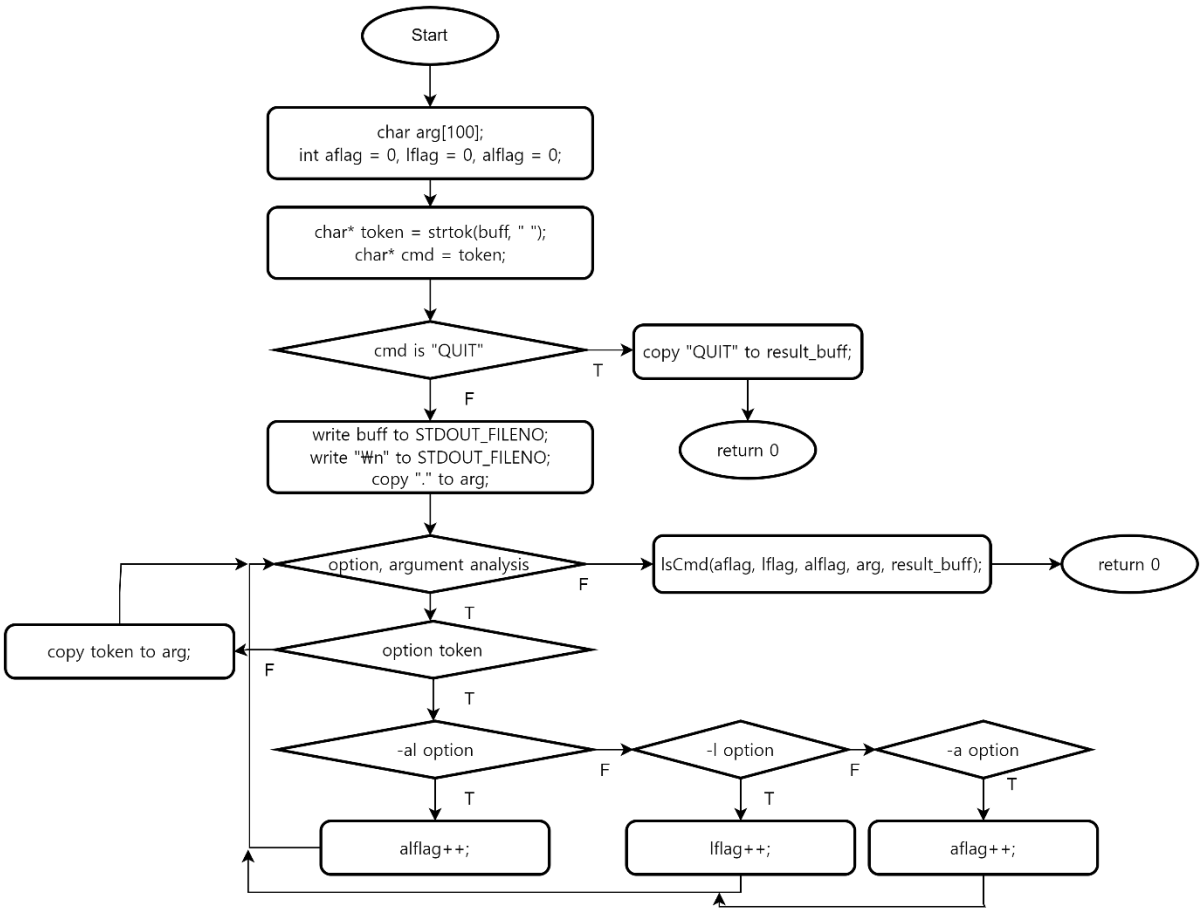
cli.c - main()


Start
char buff[MAX_BUFF], cmd_buff[MAX_BUFF],
rcv_buff[RCV_BUFF];
int n;
Open socket and connect to server
return -1
Set server address
Connect to server
return -1
Read user command
return 0
convert ls to NLST
exit(1)
FTP cmd send to server
exit(1)
read server result
exit(1)
program quit
exit(1)
Display result
return 0

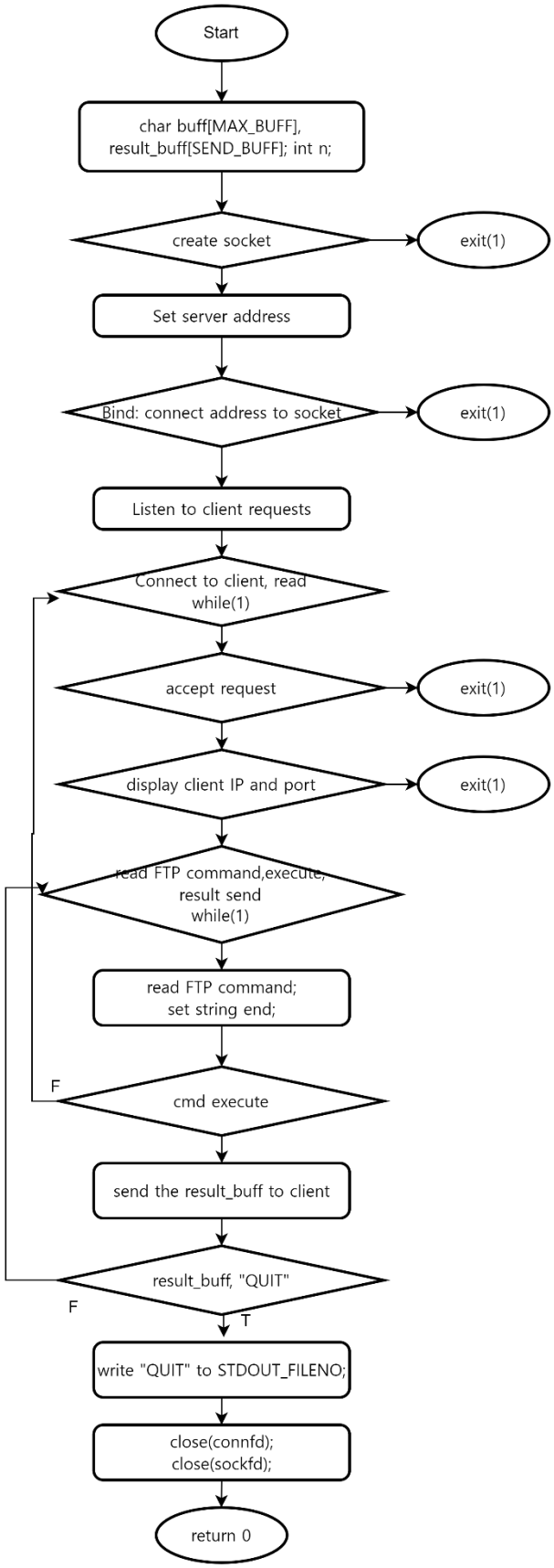
srv.c – client_info



srv.c – cmd_process



srv.c – main()


Start
char buff[MAX_BUFF],
result_buff[SEND_BUFF]; int n;
create socket
exit(1)
Set server address
Bind: connect address to socket
exit(1)
Listen to client requests
Connect to client, read
while(1)
accept request
exit(1)
display client IP and port
exit(1)
read FTP command,execute,
result send
while(1)
read FTP command;
set string end;
F
cmd execute
send the result_buff to client
result_buff, "QUIT"
F
T
write "QUIT" to STDOUT_FILENO;
close(connfd);
close(sockfd);
return 0

Pseudo code

cli.c

```
int conv_cmd(char* buff, char*cmd_buff) {
    char *token = strtok(buff, " ");                    // user command
    char *userCmd = token;
    if userCmd is "quit" then {
        copy "QUIT" to cmd_buff;                        // FTP command
        set string end; }
    else if userCmd is "ls" then {
        copy "NLST" to cmd_buff;                        // FTP command
        while((token = strtok(NULL, " "))!=NULL){ // option, argument parsing
            add a space to cmd_buff;
            add token to cmd_buff; }
        integer n = length of cmd_buff;
        set string end; }
    else {
        write "Error: invalid command" to STDERR_FILENO;
        return -1; }
    return 1
}
```

```
int main(integer argc, array argv) {
    char buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[RCV_BUFF];
    int n;              // cmd_buff length
    // Open socket and connect to server
    int sockfd, len; struct sockaddr_in server_addr; char haddr; haddr = argv[1];
    int PORTNO = atoi(argv[2]); // port
    if sockfd = socket(PF_INET, SOCK_STREAM, 0) < 0 then // create socket
```

```
        print "Error: can't create socket.";
        return -1;
// Set server address
set buffer to 0;
set server_addr to 0;
server_addr.sin_family = AF_INET;                // set family
server_addr.sin_addr.s_addr = inet_addr(haddr); // set IP address
server_addr.sin_port = htons(PORTNO);        // set port
// Connect to server
if connect(sockfd, server_addr, sizeof(server_addr)) < 0 then
        print "Error: can't connect.";
        return -1;
// Read user command and send to server, receive results and print
write "> " to STDOUT_FILENO // user input
while len = read(STDIN_FILENO, buff, sizeof(buff)) > 0 { // read user command line
        clear newline characters;
        if newline != NULL then
                set string termination;
        if conv_cmd(buff, cmd_buff) < 0 then // convert ls to NLST (buff->cmd_buff)
                print "Error: conv_cmd() error!!";
                exit(1);
        n = length of cmd_buff;
        if write(sockfd, cmd_buff, n) != n then   // FTP cmd send to server
                print "Error: write() error!!";
                exit(1);
        if read(sockfd, rcv_buff, RCV_BUFF-1) < 0 then // read server result
                print "Error: read() error!!";
                exit(1)
```

```
        set string termination;
        if not strcmp(rcv_buff, "QUIT") then       // program quit
            print "Program quit!!";
            exit(1);
        // Display result
        write rcv_buff to STDOUT_FILENO;
        write "₩n" to STDOUT_FILENO;
        write "> " to STDOUT_FILENO;
    }     //end while
    return 0;
}
```

srv.c

```
int client_info(pointer client_addr) {
    char ipAddr[20]; // ip address
    inet_ntop(AF_INET, &(client_addr->sin_addr), ipAddr, 20) // Convert to dot ip format
    print "==========Client info==========";
    print "client IP:", ipAddr;
    print "client port:", ntohs(client_addr->sin_port);
    print "================================";
    return 0;
}
```

```
int cmd_process(string buff, string result_buff)
    char arg[100];                          // ls argument (path)
    int aflag = 0, lflag = 0, alflag = 0; // ls options (-a, -l, -al)
    char* token = strtok(buff, " "); // command
    char* cmd = token;
    if cmd is "QUIT" then { // quit command
```

```
        copy "QUIT" to result_buff;
        return 0; }
    else {                          // ls command execute
        write buff to STDOUT_FILENO;
        write "₩n" to STDOUT_FILENO;
        copy "." to arg;        // set default path to current directory
        while token = strtok(NULL, " ") is not NULL {         // option, argument analysis
            if token[0] is '-' then {                                   // option
                if token[2] is 'l' then increment alflag;               //-al
                else if token[1] is 'l' then increment lflag;           //-l
                else if token[1] is 'a' then increment aflag; }         //-a
            else   {                                                // no option, argument = path
                copy token to arg; }
        lsCmd(aflag, lflag, alflag, arg, result_buff);                  // ls execute
    return 0
}
```

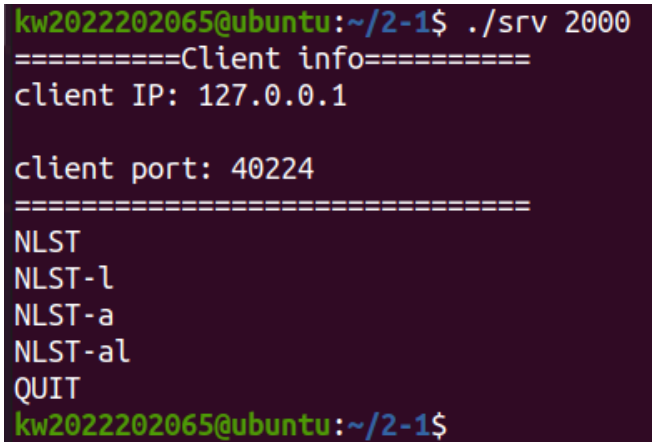
```
int main(integer argc, array argv) {
    char buff[MAX_BUFF], result_buff[SEND_BUFF]; int n;
    // Open socket and listen
    struct sockaddr_in server_addr, client_addr;
    int sockfd, connfd;
    int len                         // client addr length;
    int port = atoi(argv[1]);
    if sockfd = socket(PF_INET, SOCK_STREAM, 0) < 0 then   {        // create socket
        write "Server: Can't open stream socket." to STDERR_FILENO;
        exit(1); }
    // Set server address
    set server_addr to 0;
```

```
server_addr.sin_family = AF_INET;                    // set family
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);     // set IP address
server_addr.sin_port = htons(port);                  // set port
// Bind: connect address to socket
if bind(sockfd, server_addr, sizeof(server_addr)) < 0 then {
    write "Server: Can't bind local address." to STDERR_FILENO;
    exit(1); }
// Listen to client requests
listen(sockfd, 5);
// Connect to client, read and execute commands
while true {
    len = sizeof(client_addr);
    connfd = accept(sockfd, client_addr, len);       // accept request
    if connfd < 0 then {
        write "Server: accept failed." to STDERR_FILENO;
        exit(1); }
    if client_info(client_addr) < 0 then {       // display client IP and port
        write "Error: client_info() error!!" to STDERR_FILENO; }
    while true {                    // read FTP command, execute, send the result to client
        n = read(connfd, buff, MAX_BUFF);                    // read FTP command
        buff[n] = '₩0';                                      // set string end
        if cmd_process(buff, result_buff) < 0 then {         // cmd execute
            write "Error: cmd_process() error!!" to STDERR_FILENO;
            break; }
        write result_buff to connfd;                         // send to client
        if not strcmp(result_buff, "QUIT") then {            // program quit
            write "QUIT" to STDOUT_FILENO;
            close(connfd);
```

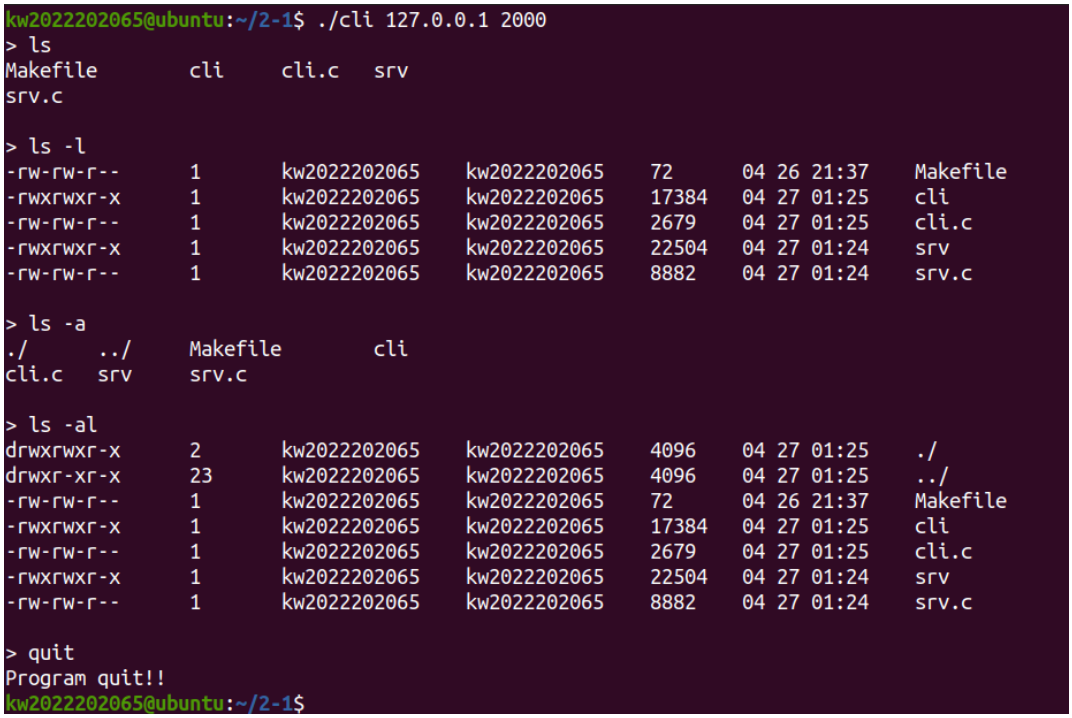
```
                    close(sockfd);
                    return 0;
                } // end if
            } //end while
        } end while
        close(connfd);
        close(sockfd);
        return 0;
}
```

결과화면

서버에서 port 를 2000 으로 입력한 뒤, 클라이언트에서 같은 port 로 연결을 했을 때 클라이언트의 ip 와 port 정보를 출력한 결과이다. 클라이언트에서 명령어를 입력할 때마다 변환된 FTP 명령어를 출력한다. QUIT 이 들어오면 서버 프로그램도 종료된다.



서버를 먼저 실행한 뒤, 클라이언트에서 ip 주소와 port 를 입력하고 명령어를 입력했을 때 출력된 결과이다. ls 명령어를 실행하며, -a, -l, -al 옵션을 구현하였다. 옵션에 따라 기본 출력, a 는 숨김파일 및 디렉토리까지 표시, l 은 파일의 정보를 자세하게 출력한다. quit 이 들어올 때까지 입력을 계속 받다가, quit 이 들어오면 메시지를 띄운 뒤 프로그램을 종료한다.



고찰

이번에 소켓의 연결 과정을 구현하면서, 여러 시행착오를 겪었다. 먼저 클라이언트를 구현하는 과정에서, 처음에 ip 와 port 를 인자로 받아야 했었다. 이때 port 를 바로 htons()를 이용하여 sin_port 로 설정하였다가 실행해보니 서버와 연결에 실패했다는 오류 메시지가 떴다. 다시 확인해보니, port 를 문자열로 받았었기 때문에 int 형으로 바꿔주어야 했다. 그래서 atoi()를 이용해 정수로 바꾼 다음 htons 를 했더니 정상적으로 전달될 수 있었다. 또한 클라이언트에서는 받은 ip 를 문자열 그대로 inet_addr 에 넣어주면 됐었는데, 서버에서는 다르게 구현해야 했다. 클라이언트의 ip 를 받아와야 해서 처음 설정할 때는 어떻게 해야 할지 몰랐었는데, INADDR_ANY 를 사용하면 된다는 것을 깨달았다. 이걸 쓰면 ip 주소를 몰라도 프로그램이 동작하며, 서버가 모든 인터페이스로 향하는 패킷을 수신할 수 있기 때문에 클라이언트의 ip 를 받아올 수 있다는 것을 알았다. 이를 이용하여 client_info()를 구현하는 부분에서도 몇몇 오류가 있었는데, 처음엔 client_addr 에서 바로 sin_addr 를 가져오면 ip 주소가 출력될 줄 알고 구현했었다. 하지만 다시 도트 형식으로 바꿔야 한다는 점을 깨닫고, inet_ntop()를 이용해서 변환하니 잘 출력될 수 있었다. 그리고 서버와 클라이언트를 연결할 때, 포트를 1000 으로 하면 실패했었다. 이에 2000 이상으로 하니 잘 실행될 수 있었다.

서버와 클라이언트의 연결을 성공적으로 마치고 나니, 이번엔 명령어가 잘못되었다는 오류가 떴었다. 하나씩 다시 확인해보다가 버퍼에서 문제가 있었다는 점을 확인했다. ls 명령어를 read 로 읽고 buff 에 저장할 때, 개행 문자가 저장되고 문자열 끝이 제대로 설정이 안돼서 그런 것이었다. 이에 명령어 라인의 끝의 개행 문자를 지우고 '₩0'으로 문자열 끝을 설정하니 명령어를 잘 해석할 수 있었다. 이와 비슷한 문제가 다른 버퍼에서도 발견되어서 대부분의 버퍼를 새로 설정할 땐 반드시 문자열 끝을 설정하는 작업을 추가하였다. 특히 ls 의 결과를 저장한 result_buff 에서도 이 문제가 나타났다. strcpy 와 strcat 으로 결과를 저장한 다음, 문자열 끝을 제대로 설정하지 않고 클라이언트에 전송하였다가 클라이언트에서 결과를 출력할 때 문자열이 잘라져서 보이고 끝까지 출력이 안되었다. 따라서 서버에서 전송하기 전에 문자열 끝을 설정해주고 보내니 클라이언트에서도 끝까지 잘 출력될 수 있었다.

이번 과제를 통해 서버와 클라이언트의 소켓 연결, 전달 과정을 직접 구현해보면서 소켓에 대해 자세하게 알 수 있었다. 터미널에서 실행할 때 그 순서가 더 잘 와닿았던 것 같다. 또한 각 함수마다 예외처리를 해주어야 코드를 완성하는 과정에서 빨리 문제점을 깨닫고 수정할 수 있다는 점도 더 잘 알 수 있었다. 이번에 코드 오류를 수정하면서 이러한 예외처리와, 버퍼 관련 문제를 다루는 것에 대한 중요성을 깨달았다.

Reference

강의자료 참고